МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №4
по дисциплине «Алгоритмы и структуры данных»
Тема: Программирование алгоритмов с бинарными деревьями
Вариант 1в

Студент гр. 8304 _____ Щука А. А.

Преподаватель _____ Фирсов М. А.

Санкт-Петербург
2019

Цель работы.

Познакомиться с такой часто используемой на практике, особенно при решении задач кодирования и поиска, нелинейной структурой данных, как бинарное дерево, способами её представления и реализации, получить навыки решения задач обработки бинарных деревьев.

Постановка задачи.

Вариант 1в: Задано бинарное дерево *b* типа *BT* с типом элементов *Elem*. Для введенной пользователем величины *E* (**var** *E*: *Elem*):

а) определить, входит ли элемент *E* в дерево *b*;

б) определить число вхождений элемента *E* в дерево *b*;

в) найти в дереве *b* длину пути (число ветвей) от корня до ближайшего узла с элементом *E* (если *E* не входит в *b*, за ответ принять –1).

Описание алгоритма.

Для начала программа должна считать данные и занести их в структуру. Дерево реализуется на базе вектора: в структуре каждого узла должен храниться индекс левого и правого узлов в массиве. Если узел в дереве пустой, то массив с данным индексом хранит нулевой элемент.

Далее для нахождения элемента в массиве необходимо сначала идти по левому поддереву, если натыкаемся на нулевой узел, то возвращаемся на шаг назад и проходим по правому поддереву. Для подсчета длины пути до ближайшего искомого элемента необходимо каждый раз, когда переходим в левое или правое поддерево увеличивать длину пути на единицу, когда возвращаемся на шаг назад нужно уменьшать длину пути на единицу. Затем сравнить каждую из получившихся длин.

Спецификация программы.

Программа предназначена для определения вхождения элемента в дерево, подсчета количества искомых элементов, нахождение длины пути до ближайшего элемента.

Программа написана на языке C++. Входными данными являются элемент и дерево в упрощенной скобочной записи. Данные вводятся либо из файла, либо с помощью GUI. Выходными данными являются промежуточные значения и конечный результат. Данные выводятся на экран монитора.

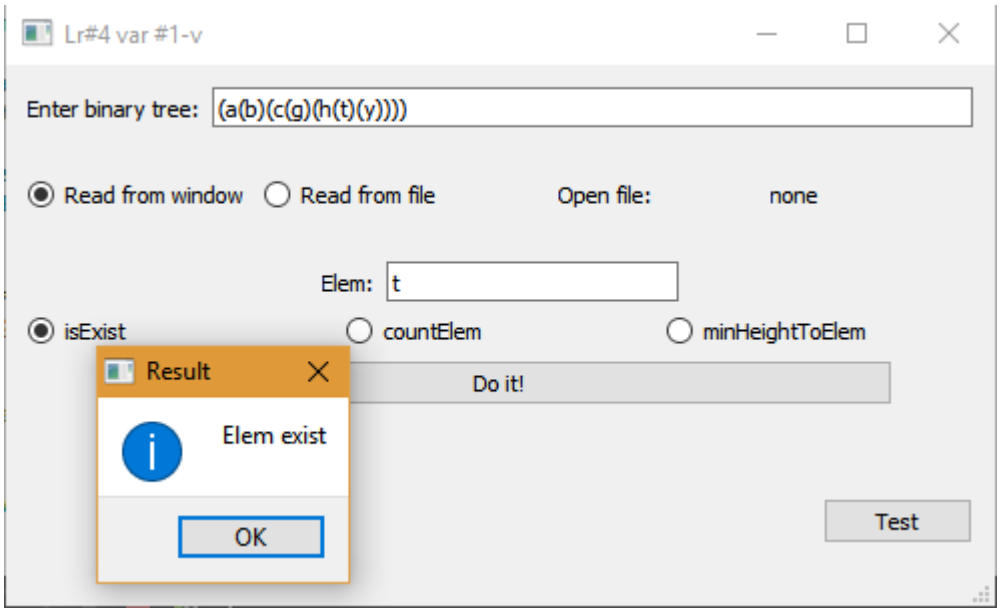

Рисунок 1 – Пример работы программы

Тестирование.

Результаты тестирования программы приведены в табл. 1.

Таблица 1 – Результаты тестирования

Ввод	Вывод
(a(b(c(d(d(e(f(d)))))))) d	Элемент найден Количество: 3 Кратчайший путь равен 3

(a(b)(c)) d	Элемент не найден
(a(b(t)(r(u)(d)))(c(k)(d))) d	Элемент найден Количество: 2 Кратчайший путь равен 2
(a(a)(b))	Элемент введен неверно
(a) a	Элемент найден Количество: 1 Кратчайший путь равен 0
() d	Не был добавлен элемент в узел Ошибка ввода
a(b)(c) c	Скобочная запись бинарного дерева должна начинаться и заканчиваться скобками Ошибка ввода
(a(b)(c)(d)) d	Веток больше, чем 2 Ошибка ввода
(a(b)(c) a	Ошибка ввода
(a(b)c)) a	Ошибка! Несколько элементов подряд без скобок

Описание функций и СД.

```
int minHeightToElem(const T& elem) const;
int _minHeightToElem(const T& elem, size_t heght = 0,
        size_t index = 0, size_t depth = 1) const;
```

Метод принимает искомый элемент и вызывает приватный метод, который принимает искомый элемент, минимальный путь, индекс вершины и глубину.

Методы возвращают значение типа *int*.

Если корень дерева совпадает с искомым элементом, функция возвращает значение глубины. Иначе функция возвращает наибольшее значение рекурсивного применения функции к поддеревьям (если они существуют). Если поддеревьев нет, функция возвращает -1.

```
bool isExistElem(const T& elem) const;
bool _isExistElem(const T& elem, size_t index = 0, size_t depth = 1) const;
```

Метод принимает искомый элемент и вызывает приватный метод, который принимает искомый элемент, индекс вершины и глубину.

Методы возвращают значение типа *bool*.

Если корень дерева совпадает с искомым элементом, функция возвращает *true*. Иначе, если у дерева есть левая и правая ветви, функция применяется к поддеревьям рекурсивно со значением глубины, увеличенным на 1. Если ветвей нет, возвращает false.

```
size_t countElem(const T& elem) const;
size_t _countElem(const T& elem, size_t index = 0, size_t depth = 1) const;
```

Метод принимает искомый элемент и вызывает приватный метод, который принимает искомый элемент, индекс вершины и глубину.

Методы возвращают значение типа *size_t*.

Если корень дерева совпадает с искомым элементом, функция возвращает 1 + (рекурсивное применение функции к левому и правому поддеревьям (если они существуют)).

Выводы.

В ходе выполнения данной лабораторной работы был реализован шаблонный класс бинарное дерево на базе вектора, а также написаны функции поиска для работы с ним.

Приложение А. Исходный код программы

main.cpp

```
#include "mainwindow.h"
#include <QApplication>

//var #1-v


int main(int argc, char *argv[])
{
    QApplication app(argc, argv);

    MainWindow window;
    window.setWindowTitle("Lr#4 var #1-v");
    window.show();

    return app.exec();
}
```

mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    file = new QFile;
    in = new QTextStream;
    charTree = new MyBinaryTree<char>;
    dir = new QDir;
    ui->readFromWindow_radioButton->setChecked(true);
    ui->isExist_radioButton->setChecked(true);
}


MainWindow::~MainWindow()
{
    delete ui;
    delete dir;
    delete file;
    delete in;
    delete charTree;
}


void MainWindow::on_test_pushButton_clicked()
{
    /*
     * Функция тестирования. Тестовые данные считываются из папки Tests
    */

    ui->readFromFile_radioButton->setChecked(true);
    qDebug() << "Тестирование.";

    std::string elemStr;
    std::string treeStr;
    dir->cd(QApplication::applicationDirPath() + "/Tests");
    QStringList listFiles = dir->entryList(QStringList("*.txt"), QDir::Files);

    for (auto fileName : listFiles) {
        if (fileName == "." || fileName == "..")
            continue;

        qDebug();
        qDebug() << "Тестовые данные из файла:" << fileName;

        file->close();
        file->setFileName(dir->path() + "/" + fileName);
        file->open(QFile::ReadOnly | QFile::Text);
        in->setDevice(file);
```

```
        while (!in->atEnd()) {
            elemStr = in->readLine().toStdString();

            bool isIsExist = false;
            bool isCountElem = false;
            bool isMinHeightToElem = false;

            if (ui->isExist_radioButton->isChecked())
                isIsExist = true;
            else if (ui->countElem_radioButton->isChecked())
                isCountElem = true;
            else
                isMinHeightToElem = true;

            if (isIsExist)
                qDebug() << "Проверка на существование элемента:";
            else if (isCountElem)
                qDebug() << "Подсчет количества вхождений элемента в дерево:";
            else
                qDebug() << "Подсчет минимального пути до элемента в дереве:";

            if (elemStr.size() != 1) {
                qDebug() << "Некорректный элемент!";
            }
            else {
                qDebug() << "Считанный элемент: " << elemStr[0];
                if (readCharTree(charTree)) {
                    qDebug() << "Считанное дерево:";
                    qDebug() << charTree->toAssignmentList().c_str();
                    if (isIsExist) {
                        if(charTree->isExistElem(elemStr[0])) {
                            qDebug() << "Элемент есть в дереве";
                        }
                        else {
                            qDebug() << "Элемента нет в дереве!";
                        }
                    }

                    if (isCountElem) {
                        size_t count = charTree->countElem(elemStr[0]);
                        std::string result = "Number of elements: ";
                        result += std::to_string(count);
                        qDebug() << "Количество элементов в дереве: " << count;
                    }

                    if (isMinHeightToElem) {
                        int minHeightToElem = charTree->minHeightToElem(elemStr[0]);
                        std::string result = "Min. height to element: ";
                        result += std::to_string(minHeightToElem);
                        qDebug() << "Минимальный путь до элемента: " << minHeightToElem;
                    }
                }
                else {
                    qDebug() << "Некорректное дерево!";
                }
            }
            charTree->clearTree();
            qDebug();
        }
    }
    ui->readFromWindow_radioButton->setChecked(true);
}


void MainWindow::on_readFromWindow_radioButton_clicked()
{
    //выбор считывания из окна
    file->close();
    ui->tree_lineEdit->setEnabled(true);
    ui->textLabel_1->setEnabled(true);
    ui->elem_lineEdit->setEnabled(true);
    ui->textlabel_2->setEnabled(true);
    ui->openFile_textLabel->setText("none");
}


void MainWindow::on_readFromFile_radioButton_clicked()
{
    //выбор считывания из файла
```

```
    file->close();

    ui->textLabel_0->setEnabled(false);
    ui->tree_lineEdit->setEnabled(false);
    ui->elem_lineEdit->setEnabled(false);
    ui->textlabel_2->setEnabled(false);
    QString fileName = QFileDialog::getOpenFileName(this,
                                                    "Open file");
    file->setFileName(fileName);
    file->open(QFile::ReadOnly | QFile::Text);
    in->setDevice(file);

    ui->openFile_textLabel->setText(fileName);
}


bool MainWindow::readCharTree(MyBinaryTree<char> *charTree)
{
    /*
     * Функция считывания дерева
    */

    std::string tmpStr = "";

    //считывание строки из файла или из окна
    if (ui->readFromFile_radioButton->isChecked()) {
        tmpStr = in->readLine().toStdString();
    }
    else {
        tmpStr = ui->tree_lineEdit->text().toStdString();
    }

    //проверка строки на корректность и создание дерева
    qDebug() << "Считанная строка:" << tmpStr.c_str();
    if (isCorrectStr(tmpStr)) {
        charTree->createCharTree(tmpStr);
        return true;
    }
    else {
        return false;
    }
}


bool MainWindow::isCorrectStr(const std::string &str)
{
    /*
     * Проверка входной строки(дерева в упрощенной скобочной записи) на
     * корректность
    */

    bool isElem = false;
    bool f = true;
    if (str[0] != '(') {
        qDebug() << "Скобочная запись бинарного дерева должна начинаться и заканчиваться скобками";
        return false;
    }
    if (str[str.length() - 1] != ')') {
        qDebug() << "Скобочная запись бинарного дерева должна начинаться и заканчиваться скобками";
        return false;
    }
    size_t indexStart;
    size_t numBrackets = 0;
    for (indexStart = 1; indexStart < str.length() - 1; indexStart++) {
        if (str[indexStart] != ' ' && str[indexStart] != '(' && str[indexStart] != ')') {
            if (!isElem) {
                isElem = true;
            }
            else {
                qDebug() << "Ошибка! Несколько элементов подряд без скобок";
                return false;
            }
        }
        if (str[indexStart] == '(') {
            if (!isElem) {
                qDebug() << "Не был добавлен эдемент в узел";
                return false;
            }
            numBrackets++;
```

```
            if (numBrackets > 2) {
                qDebug() << "Веток больше, чем 2";
                return false;
            }
            size_t indexEnd = indexStart;
            int openB = 1;
            int closeB = 0;
            while (openB > closeB) {
                indexEnd++;
                if (indexEnd >= str.length()) {
                    qDebug() << "Некорректный ввод строки";
                    return false;
                }
                if (str[indexEnd] == '(') {
                    openB++;
                }
                else if (str[indexEnd] == ')') {
                    closeB++;
                }
            }
            f = f && isCorrectStr(str.substr(indexStart, indexEnd - indexStart + 1));
            indexStart = indexEnd;
        }
    }
    if (str[indexStart] == ')') {
        if (!isElem) {
            qDebug() << "Не был добавлен элемент в узел";
            return false;
        }
        if (indexStart == str.length() - 1) {
            return f;
        }
        else {
            qDebug() << "Некорректный ввод";
            return false;
        }
    }
    return false;
}


void MainWindow::on_do_pushButton_clicked()
{
    bool isIsExist = false;
    bool isCountElem = false;
    bool isMinHeightToElem = false;

    if (ui->isExist_radioButton->isChecked())
        isIsExist = true;
    else if (ui->countElem_radioButton->isChecked())
        isCountElem = true;
    else
        isMinHeightToElem = true;

    if (isIsExist)
        qDebug() << "Проверка на существование элемента:";
    else if (isCountElem)
        qDebug() << "Подсчет количества вхождений элемента в дерево:";
    else
        qDebug() << "Подсчет минимального пути до элемента в дереве:";


    std::string elem;
    if (ui->readFromWindow_radioButton->isChecked()) {
        elem = ui->elem_lineEdit->text().toStdString();
    }
    else {
        elem = in->readLine().toStdString();
    }

    if (elem.size() != 1) {
        qDebug() << "Некорректный элемент!";
        QMessageBox::critical(this, "Error", "Incorrect elem!");
    }
    else {
        qDebug() << "Считанный элемент: " << elem[0];
        if (readCharTree(charTree)) {
            qDebug() << "Дерево: ";
            qDebug() << charTree->toAssignmentList().c_str();
```

```
            if (isIsExist) {
                if(charTree->isExistElem(elem[0])) {
                    qDebug() << "Элемент есть в дереве";
                    QMessageBox::information(this, "Result", "Elem exist");
                }
                else {
                    qDebug() << "Элемента нет в дереве!";
                    QMessageBox::warning(this, "Result", "Elem doesn't exist");
                }
            }

            if (isCountElem) {
                size_t count = charTree->countElem(elem[0]);
                std::string result = "Number of elements: ";
                result += std::to_string(count);
                qDebug() << "Количество элементов в дереве: " << count;
                QMessageBox::information(this, "Result",  result.c_str());
            }

            if (isMinHeightToElem) {
                int minHeightToElem = charTree->minHeightToElem(elem[0]);
                std::string result = "Min. height to element: ";
                result += std::to_string(minHeightToElem);
                qDebug() << "Минимальный путь до элемента: " << minHeightToElem;
                QMessageBox::information(this, "Result",  result.c_str());
            }
        }
        else {
            qDebug() << "Некорректное дерево!";
            QMessageBox::critical(this, "Error", "Incorrect tree!");
        }
    }
    charTree->clearTree();
}
```

mybinarytree.h

```
#ifndef MYBINARYTREE_H
#define MYBINARYTREE_H

#include <QDebug>
#include <QObject>
#include <iostream>
#include <ostream>
#include <memory>


constexpr size_t SIZE = 500;

template <class T>
class MyBinaryTree
{
    /* Класс-реализация бинарного дерева на массиве.
     * Содержит массив элементов дерева и размер дерева.
    */

public:
#pragma pack(2)
    //Структура элемента дерева
    struct Node
    {
        int leftElem;
        int rightElem;
        bool isNull;
        T data;
    };
    typedef std::shared_ptr<Node> NodeP;
#pragma pack()

protected:
    NodeP binTree[SIZE];
    size_t size;

public:
    explicit MyBinaryTree();
    ~MyBinaryTree() = default;
    void clearTree();
```

```
    MyBinaryTree& operator=(const MyBinaryTree& tree) = delete;
    MyBinaryTree(const MyBinaryTree&) = delete;

    //Getters-методы
    bool isNull(size_t index) const;
    int getLeft(size_t index) const;
    int getRight(size_t index) const;
    size_t getSize() const;
    const T& getElem(size_t index) const;

    //Методы для решения подзадач. Шаблон метода для создания дерева реализован только для char
    size_t createCharTree(const std::string& expression) = delete;
    int minHeightToElem(const T& elem) const;
    bool isExistElem(const T& elem) const;
    size_t countElem(const T& elem) const;

    std::string toStdString() const;
    std::string toAssignmentList() const;

protected:
    //метод для создания элемента
    void createNode();
    //метод для преобразования элемента к строке
    void NodeToStdString(const MyBinaryTree* tree, NodeP subTree, std::string& str) const;
    //метод для получения индекса в строке, где заканчивается поддерево
    size_t getEndIndexSubTree(const std::string& str, size_t indexEnd) const;
    //метод для преобразования дерева к уступчатому списку
    void _toAssignmentList(std::string& str, size_t index, size_t depth = 1) const;

    //приватные методы для подзадач, принимают на вход доп.параметры
    int _minHeightToElem(const T& elem, size_t heght = 0,
                         size_t index = 0, size_t depth = 1) const;
    bool _isExistElem(const T& elem, size_t index = 0, size_t depth = 1) const;
    size_t _countElem(const T& elem, size_t index = 0, size_t depth = 1) const;
};


template<class T>
MyBinaryTree<T>::MyBinaryTree()
{
    /*
     * Создание пустого дерева
    */

    this->size = 0;
}


template<typename T>
void MyBinaryTree<T>::clearTree()
{
    /*
     * Полное удаление дерева
    */

    for (size_t i = 0; i < size; ++i) {
        binTree[i].reset();
    }
    size = 0;
}


template <class T>
void MyBinaryTree<T>::createNode()
{
    /*
     * Создание элемента дерева.
     * Метод приватный т.к создание элементов должно происходить при создании дерева,
     * из-за особенностей реализации на массиве
    */

    NodeP tmp = NodeP(new Node);
    tmp->isNull = true;
    tmp->leftElem = -1;
    tmp->rightElem = -1;

    this->binTree[size] = tmp;
    size += 1;
```

```
}


template<typename T>
bool MyBinaryTree<T>::isNull(size_t index) const
{
    /*
     * Метод возвращает bool-значение, является ли элемент дерева нулевым
    */

    if (index < size) {
        return binTree[index]->isNull;
    }
    else {
        qDebug() << "Error: isNull(index >= size)";
        return false;
    }
}


template<class T>
int MyBinaryTree<T>::getLeft(size_t index) const
{
    /*
     * Метод возвращает индекс левого поддерева
    */

    if (!this->isNull(index)) {
        return this->binTree[index]->leftElem;
    }
    else {
        qDebug() << "Error: getLeft(null)";
        return -1;
    }
}


template<class T>
int MyBinaryTree<T>::getRight(size_t index) const
{
    /*
     * Метод возвращает индекс правого поддерева
    */

    if (!this->isNull(index)) {
        return binTree[index]->rightElem;
    }
    else {
        qDebug() << "Error: getRight(null)";
        return -1;
    }
}


template<class T>
size_t MyBinaryTree<T>::getSize() const
{
    /*
     * Метод возвращает размер дерева
     */

    return this->size;
}


template<class T>
const T& MyBinaryTree<T>::getElem(size_t index) const
{
    /*
     * Метод возвращает значение элемента по индексу
    */

    if (!this->isNull(index)) {
        return this->binTree[index]->data;
    }
    else {
        qDebug() << "Error: getElem(null)";
        exit(1);
    }
```

```
}


template <> inline
size_t MyBinaryTree<char>::createCharTree(const std::string& str)
{
    /*
     * Метод создания дерева. Строка должна быть корректна
     */

    size_t sizeRoot = this->size;
    this->createNode();
    char elem = 0;

    size_t indexStart = 1;
    while (str[indexStart] != '(' && str[indexStart] != ')') {
        if (str[indexStart] != ' ' && str[indexStart] != '(' && str[indexStart] != ')') {
            elem = str[indexStart];
            this->binTree[sizeRoot]->data = elem;
            this->binTree[sizeRoot]->isNull = false;
        }
        indexStart++;
    }

    if (str[indexStart] == ')') {
        return sizeRoot;
    }

    size_t indexEnd = indexStart + 1;
    indexEnd = getEndIndexSubTree(str, indexEnd);
    size_t leftTree = createCharTree(str.substr(indexStart, indexEnd - indexStart));
    this->binTree[sizeRoot]->leftElem = static_cast<int>(leftTree);

    indexStart = indexEnd;
    while (str[indexStart] != '(' && str[indexStart] != ')') {
        indexStart++;
    }

    if (str[indexStart] == ')') {
        return sizeRoot;
    }

    indexEnd = indexStart + 1;
    indexEnd = getEndIndexSubTree(str, indexEnd);
    size_t rightTree = createCharTree(str.substr(indexStart, indexEnd - indexStart));

    this->binTree[sizeRoot]->rightElem = static_cast<int>(rightTree);
    return sizeRoot;
}


template<class T>
size_t MyBinaryTree<T>::getEndIndexSubTree(const std::string &str, size_t indexEnd) const
{
    /*
     * Метод возвращает индекс в строке конца поддерева
    */

    size_t openB = 1;
    size_t closeB = 0;
    while (openB != closeB) {
        if (str[indexEnd] == '(') {
            openB++;
        }
        else if (str[indexEnd] == ')') {
            closeB++;
        }
        indexEnd++;
    }
    return indexEnd;
}


template<class T>
void MyBinaryTree<T>::_toAssignmentList(std::string &str, size_t index, size_t depth) const
{
    for (size_t i = 0; i < depth; ++i) {
        str += " ";
    }
```

```
    str += getElem(index);
    str += "\n";

    if (getLeft(index) != -1) {
        _toAssignmentList(str, getLeft(index), depth + 1);
        if (getRight(index) != -1) {
            _toAssignmentList(str, getRight(index), depth + 1);
        }
    }
}


template<class T>
std::string MyBinaryTree<T>::toStdString() const
{
    /*
     * Метод возвращает представление бинарного дерева в виде упрощенной скобочной записи
     * Элемент должен оиметь перегрузку опрератора += std::string
    */

    if (size == 0)
        return "()";

    std::string tree;
    tree += "(";
    tree += this->binTree[0]->data;
    if (this->binTree[0]->leftElem != -1) {
        NodeToStdString(this, this->binTree[this->binTree[0]->leftElem], tree);
    }

    if (this->binTree[0]->rightElem != -1) {
        NodeToStdString(this, this->binTree[this->binTree[0]->rightElem], tree);
    }
    tree += ")";
    return tree;
}


template<class T>
std::string MyBinaryTree<T>::toAssignmentList() const
{
    if (size == 0) {
        return "(empty)";
    }

    std::string str;
    str += getElem(0);
    str += "\n";
    if (getLeft(0) != -1)
        _toAssignmentList(str, getLeft(0));
    if (getRight(0) != -1)
        _toAssignmentList(str, getRight(0));

    return str;
}


template<class T>
void MyBinaryTree<T>::NodeToStdString(const MyBinaryTree* tree, MyBinaryTree::NodeP subTree,
std::string &str) const
{
    /*
     * Метода записывает в строку-результат представление элемента дерева в упрощенной
     * скобочной записи
    */

    if (!subTree->isNull) {
        str += "(";
        str += subTree->data;
        if (subTree->leftElem != -1) {
            NodeToStdString(tree, tree->binTree[subTree->leftElem], str);
        }

        if (subTree->rightElem != -1) {
            NodeToStdString(tree, tree->binTree[subTree->rightElem], str);
        }
        str += ')';
    }
    else {
```

```
        str += "()";
    }
}


template<class T>
bool MyBinaryTree<T>::isExistElem(const T& elem) const
{
    /*
     * Метод для определения существования элемента в дереве, возвращает bool
    */

    if (size < 1) {
        return false;
    }
    else {
        return _isExistElem(elem);
    }
}


template<class T>
size_t MyBinaryTree<T>::countElem(const T &elem) const
{
    /*
    * Метод для подсчета количества вхождение элемента в дереве, возвращает size_t
   */

    if (size < 1) {
        return 0;
    }
    else {
        return _countElem(elem);
    }
}


template<typename T>
int MyBinaryTree<T>::minHeightToElem(const T &elem) const
{
    /*
     * Метод для определения минимального пути до
     * элемента в дереве, возвращает -1, если элемента в дереве нет
    */
    if (size < 1) {
        return -1;
    }
    else {
        return _minHeightToElem(elem);
    }
}


template<typename T>
bool MyBinaryTree<T>::_isExistElem(const T &elem, size_t index, size_t depth) const
{
    std::string dbgStr;
    for (size_t i = 0; i < depth; ++i) {
        dbgStr += " ";
    }

    bool isLeftTreeExistElem = false;
    bool isRightTreeExistElem = false;
    if (this->isNull(index)) {
        return false;
    }
    else if (index < this->size) {
        qDebug() << dbgStr.c_str() << "Проверяется элемент:" << getElem(index);
        if (this->getElem(index) == elem)
            return true;

        if (this->getLeft(index) != -1)
            isLeftTreeExistElem = this->_isExistElem(elem, this->getLeft(index), depth + 1);

        if(this->getRight(index) != -1)
            isRightTreeExistElem= this->_isExistElem(elem, this->getRight(index), depth + 1);

        return isLeftTreeExistElem || isRightTreeExistElem;
    }
```

```
    else {
        qDebug() << "Error: isExistElem(index >= size)";
        return false;
    }
}


template<typename T>
size_t MyBinaryTree<T>::_countElem(const T &elem, size_t index, size_t depth) const
{
    std::string dbgStr;
    for (size_t i = 0; i < depth; ++i) {
        dbgStr += " ";
    }

    size_t count = 0;
    if (this->isNull(index)) {
        return 0;
    }
    else if (index < this->size) {
        qDebug() << dbgStr.c_str() << "Проверяется элемент:" << getElem(index);
        if (this->getElem(index) == elem)
            count += 1;

        if (this->getLeft(index) != -1)
            count += this->_countElem(elem, this->getLeft(index), depth + 1);

        if(this->getRight(index) != -1)
            count += this->_countElem(elem, this->getRight(index), depth + 1);

        return count;
    }
    else {
        qDebug() << "Error: countElem(index >= size)";
        return 0;
    }
}


template<typename T>
int MyBinaryTree<T>::_minHeightToElem(const T &elem, size_t height,
                                      size_t index, size_t depth) const
{
    std::string dbgStr;
    for (size_t i = 0; i < depth; ++i) {
        dbgStr += " ";
    }

    if (isNull(index)) {
        return -1;
    }
    qDebug() << dbgStr.c_str() << "Проверяется элемент:" << getElem(index);
    if (binTree[index]->data == elem) {
        qDebug() << dbgStr.c_str() << "Длинна текущего пути:" << depth - 1;
        return static_cast<int>(height);
    }

    int leftIndex = binTree[index]->leftElem;
    int rightIndex = binTree[index]->rightElem;

    int minHeightLeft = -1;
    int minHeightRight = -1;

    if (leftIndex != -1) {
        minHeightLeft = _minHeightToElem(elem, height + 1, leftIndex, depth + 1);
    }

    if (rightIndex != -1) {
        minHeightRight = _minHeightToElem(elem, height + 1, rightIndex, depth + 1);
    }

    if (minHeightLeft == -1 && minHeightRight == -1) {
        return -1;
    }
    else if (minHeightLeft != -1 && minHeightRight == -1) {
        return minHeightLeft;
    }
    else if (minHeightRight != -1 && minHeightLeft == -1) {
        return minHeightRight;
```

```
    }
    else {
        return std::min(minHeightLeft, minHeightRight);
    }
}

#endif // MYBINARYTREE_H
```

mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QTextStream>
#include <QFile>
#include <QFileDialog>
#include <QMessageBox>
#include <QDebug>
#include <QDir>
#include <QString>
#include <string>
#include "mybinarytree.h"


namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private slots:
    void on_readFromFile_radioButton_clicked();
    void on_readFromWindow_radioButton_clicked();

    bool readCharTree(MyBinaryTree<char>* charTree);
    bool isCorrectStr(const std::string& str);

    void on_test_pushButton_clicked();
    void on_do_pushButton_clicked();

private:
    Ui::MainWindow *ui;
    QTextStream* in;
    QFile* file;
    QDir* dir;
    MyBinaryTree<char>* charTree;
};

#endif // MAINWINDOW_H
```

mainwindow.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>MainWindow</class>
 <widget class="QMainWindow" name="MainWindow">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>494</width>
    <height>270</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>MainWindow</string>
  </property>
  <widget class="QWidget" name="centralWidget">
   <layout class="QVBoxLayout" name="verticalLayout">
```

```
<item>
 <layout class="QHBoxLayout" name="horizontalLayout_2">
  <item>
   <widget class="QLabel" name="textLabel_0">
    <property name="text">
     <string>Enter binary tree:</string>
    </property>
   </widget>
  </item>
  <item>
   <widget class="QLineEdit" name="tree_lineEdit"/>
  </item>
 </layout>
</item>
<item>
 <layout class="QHBoxLayout" name="horizontalLayout">
  <item>
   <widget class="QRadioButton" name="readFromWindow_radioButton">
    <property name="text">
     <string>Read from window</string>
    </property>
    <attribute name="buttonGroup">
     <string notr="true">buttonGroup</string>
    </attribute>
   </widget>
  </item>
  <item>
   <widget class="QRadioButton" name="readFromFile_radioButton">
    <property name="text">
     <string>Read from file</string>
    </property>
    <attribute name="buttonGroup">
     <string notr="true">buttonGroup</string>
    </attribute>
   </widget>
  </item>
  <item>
   <spacer name="horizontalSpacer_2">
    <property name="orientation">
     <enum>Qt::Horizontal</enum>
    </property>
    <property name="sizeType">
     <enum>QSizePolicy::Minimum</enum>
    </property>
    <property name="sizeHint" stdset="0">
     <size>
      <width>40</width>
      <height>20</height>
     </size>
    </property>
   </spacer>
  </item>
  <item>
   <widget class="QLabel" name="textLabel_1">
    <property name="text">
     <string>Open file:</string>
    </property>
   </widget>
  </item>
  <item>
   <widget class="QLabel" name="openFile_textLabel">
    <property name="text">
     <string>none</string>
    </property>
   </widget>
  </item>
 </layout>
</item>
<item>
 <layout class="QHBoxLayout" name="horizontalLayout_4">
  <item>
   <spacer name="horizontalSpacer">
    <property name="orientation">
     <enum>Qt::Horizontal</enum>
    </property>
    <property name="sizeHint" stdset="0">
     <size>
      <width>40</width>
      <height>20</height>
```

```
     </size>
    </property>
   </spacer>
  </item>
  <item>
   <widget class="QLabel" name="textlabel_2">
    <property name="text">
     <string>Elem:</string>
    </property>
   </widget>
  </item>
  <item>
   <widget class="QLineEdit" name="elem_lineEdit"/>
  </item>
  <item>
   <spacer name="horizontalSpacer_5">
    <property name="orientation">
     <enum>Qt::Horizontal</enum>
    </property>
    <property name="sizeHint" stdset="0">
     <size>
      <width>40</width>
      <height>20</height>
     </size>
    </property>
   </spacer>
  </item>
 </layout>
</item>
<item>
 <layout class="QHBoxLayout" name="horizontalLayout_6">
  <item>
   <widget class="QRadioButton" name="isExist_radioButton">
    <property name="text">
     <string>isExist</string>
    </property>
    <attribute name="buttonGroup">
     <string notr="true">buttonGroup_2</string>
    </attribute>
   </widget>
  </item>
  <item>
   <widget class="QRadioButton" name="countElem_radioButton">
    <property name="text">
     <string>countElem</string>
    </property>
    <attribute name="buttonGroup">
     <string notr="true">buttonGroup_2</string>
    </attribute>
   </widget>
  </item>
  <item>
   <widget class="QRadioButton" name="minHeightToElem_radioButton">
    <property name="text">
     <string>minHeightToElem</string>
    </property>
    <attribute name="buttonGroup">
     <string notr="true">buttonGroup_2</string>
    </attribute>
   </widget>
  </item>
 </layout>
</item>
<item>
 <layout class="QHBoxLayout" name="horizontalLayout_3">
  <item>
   <spacer name="horizontalSpacer_3">
    <property name="orientation">
     <enum>Qt::Horizontal</enum>
    </property>
    <property name="sizeType">
     <enum>QSizePolicy::Minimum</enum>
    </property>
    <property name="sizeHint" stdset="0">
     <size>
      <width>40</width>
      <height>20</height>
     </size>
    </property>
```

```
     </spacer>
    </item>
    <item>
     <widget class="QPushButton" name="do_pushButton">
      <property name="text">
       <string>Do it!</string>
      </property>
     </widget>
    </item>
    <item>
     <spacer name="horizontalSpacer_4">
      <property name="orientation">
       <enum>Qt::Horizontal</enum>
      </property>
      <property name="sizeType">
       <enum>QSizePolicy::Minimum</enum>
      </property>
      <property name="sizeHint" stdset="0">
       <size>
        <width>40</width>
        <height>20</height>
       </size>
      </property>
     </spacer>
    </item>
   </layout>
  </item>
  <item>
   <spacer name="verticalSpacer">
    <property name="orientation">
     <enum>Qt::Vertical</enum>
    </property>
    <property name="sizeType">
     <enum>QSizePolicy::Minimum</enum>
    </property>
    <property name="sizeHint" stdset="0">
     <size>
      <width>20</width>
      <height>40</height>
     </size>
    </property>
   </spacer>
  </item>
  <item>
   <layout class="QHBoxLayout" name="horizontalLayout_5">
    <item>
     <spacer name="horizontalSpacer_6">
      <property name="orientation">
       <enum>Qt::Horizontal</enum>
      </property>
      <property name="sizeHint" stdset="0">
       <size>
        <width>40</width>
        <height>20</height>
       </size>
      </property>
     </spacer>
    </item>
    <item>
     <widget class="QPushButton" name="test_pushButton">
      <property name="text">
       <string>Test</string>
      </property>
     </widget>
    </item>
   </layout>
  </item>
 </layout>
</widget>
<widget class="QMenuBar" name="menuBar">
 <property name="geometry">
  <rect>
   <x>0</x>
   <y>0</y>
   <width>494</width>
   <height>21</height>
  </rect>
 </property>
</widget>
```

```
  <widget class="QStatusBar" name="statusBar"/>
 </widget>
 <layoutdefault spacing="6" margin="11"/>
 <resources/>
 <connections/>
 <buttongroups>
  <buttongroup name="buttonGroup"/>
  <buttongroup name="buttonGroup_2"/>
 </buttongroups>
</ui>
```